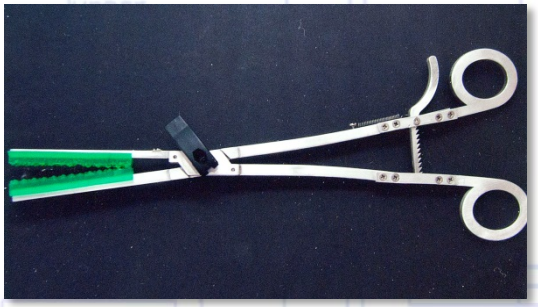
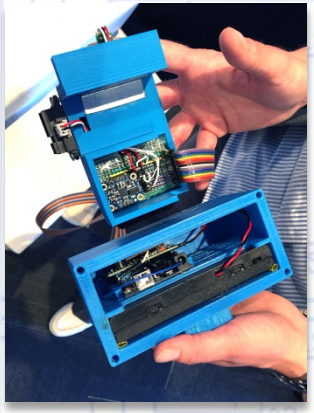
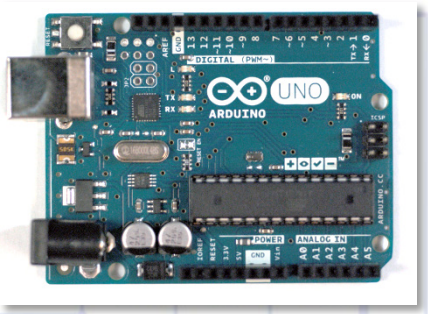
Introductory Medical Device Prototyping

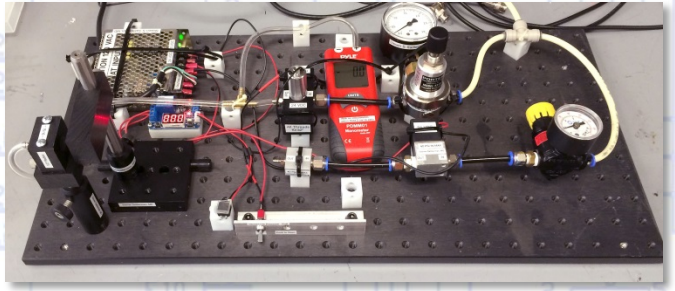
Arduino Part 1

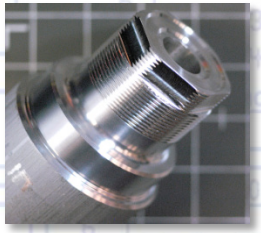
Prof. Steven S. Saliterman, http://saliterman.umn.edu/
Department of Biomedical Engineering, University of Minnesota









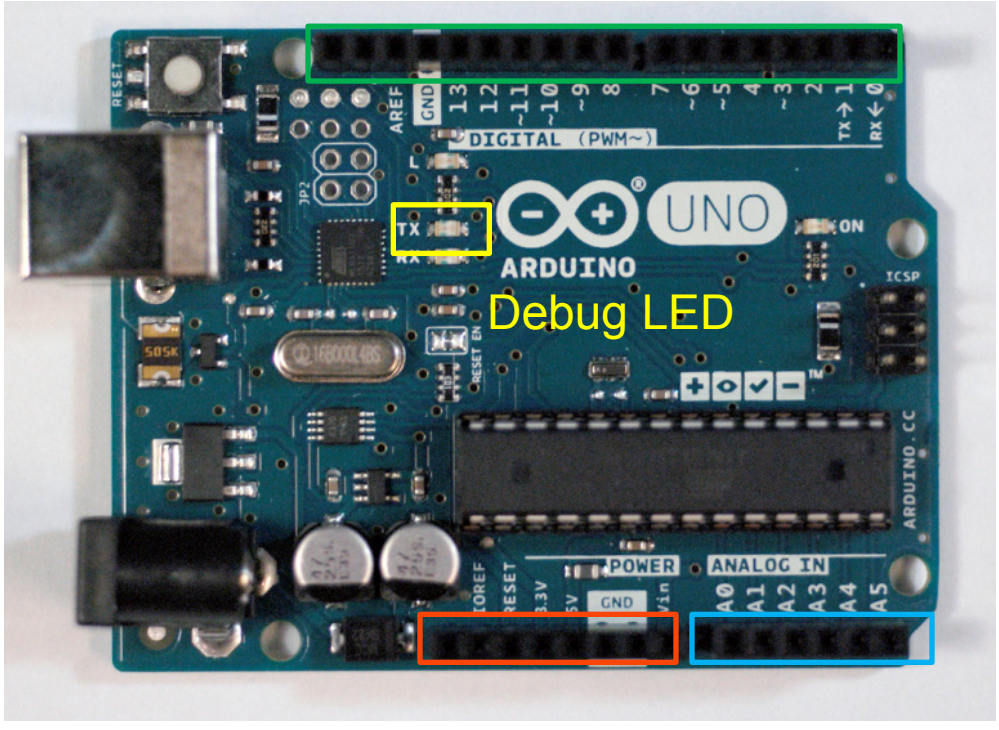


Power & Interface…

Reset Button

USB Interface

7 to 12 VDC Input



Digital I/O Pins

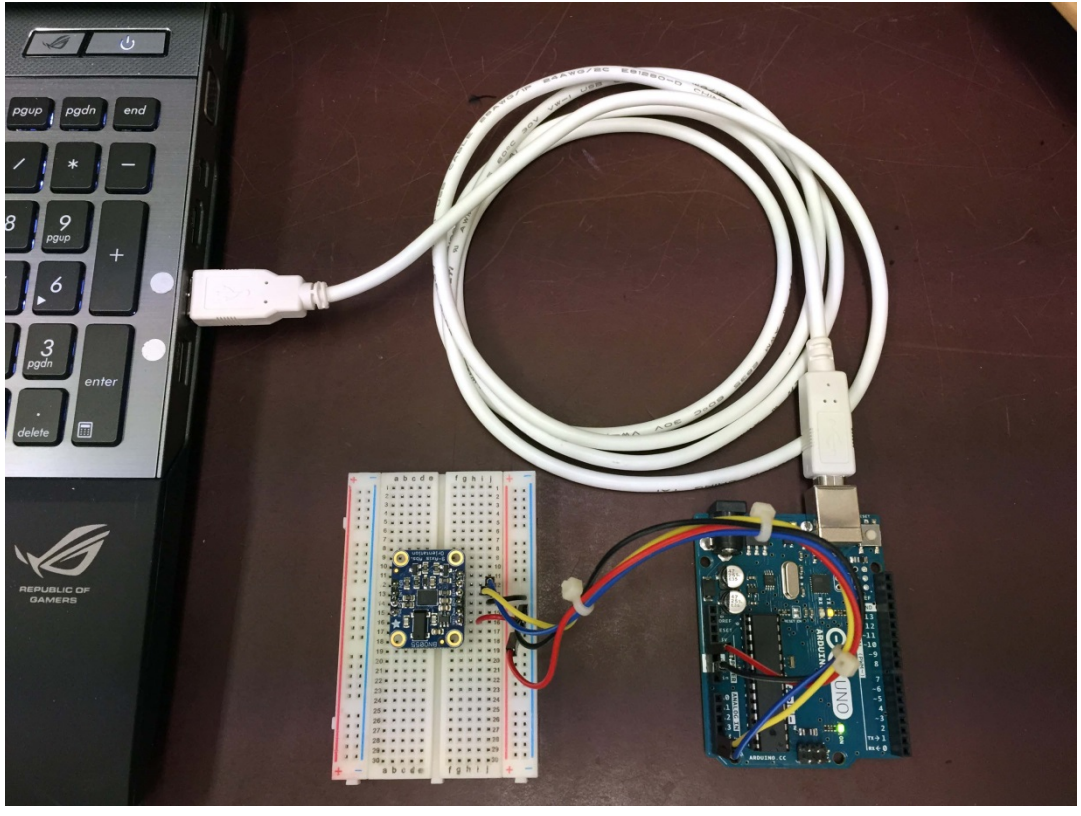
ICSP Connector

ATMega Microcontroller

Power & Auxiliary Pins

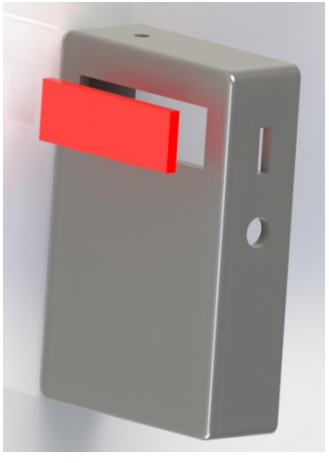
Analog –to - Digital Converter Pins

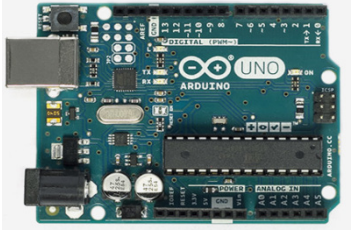
USB Connection to Computer…



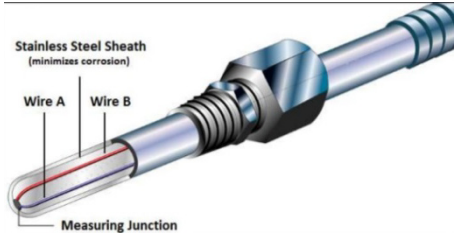
Consider this Device Concept

Task: Make a digital thermometer consisting of an enclosure, microcontroller board, thermocouple sensor, digital display, sound alert, slide switch, pushbutton and battery…

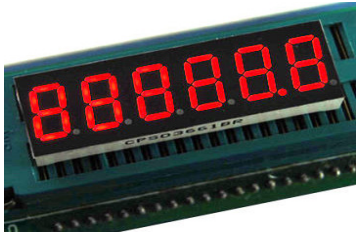




http://store-usa.arduino



http://www.thermometricscorp.com

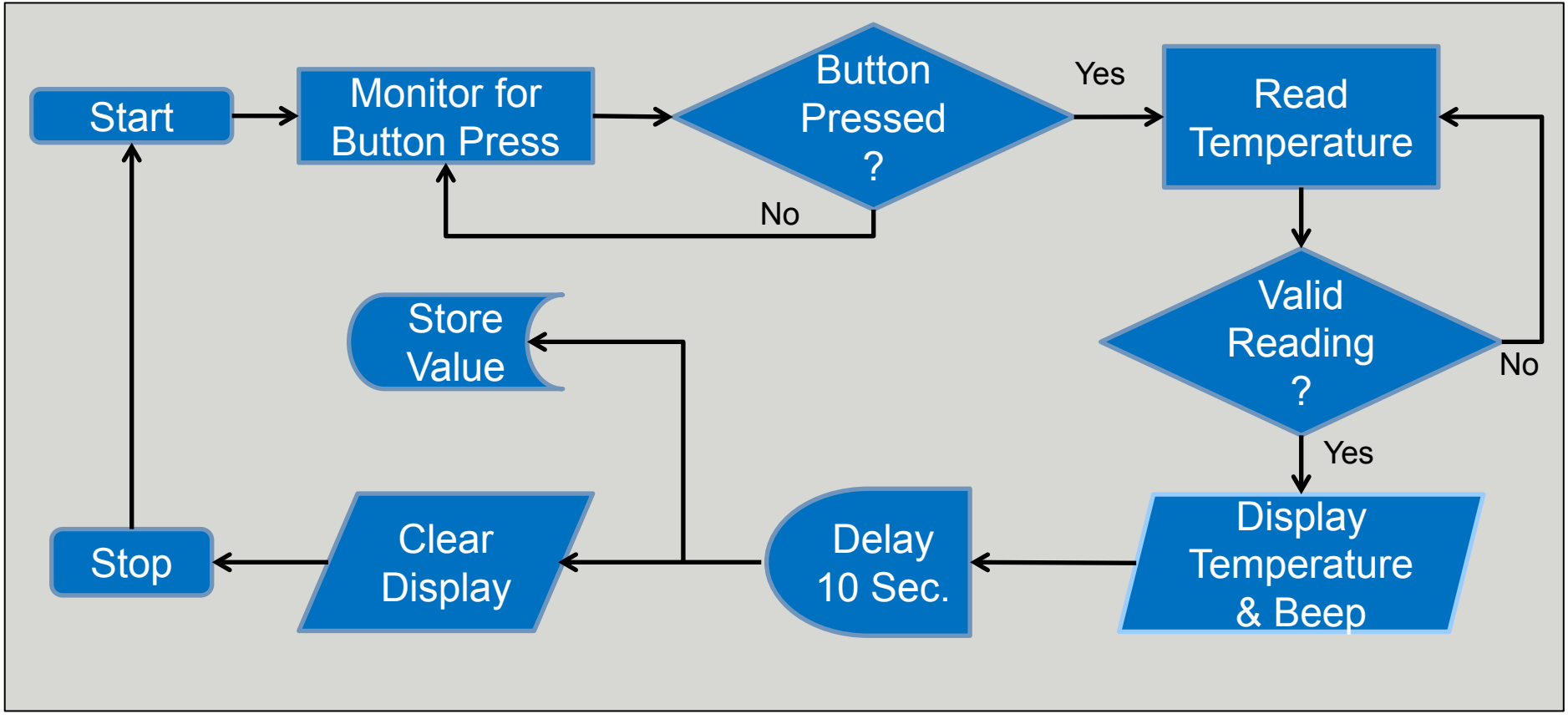


http://www.globalsources.com

Formulate an Algorithm…

1. When the pushbutton is pressed…
   - Measure the temperature,
   - Beep when the reading is good,
   - Display the value for ten seconds, and finally
   - Save the value to memory.
2. Start all over again.
3. *Now flowchart this…*

Flowchart the Algorithm…



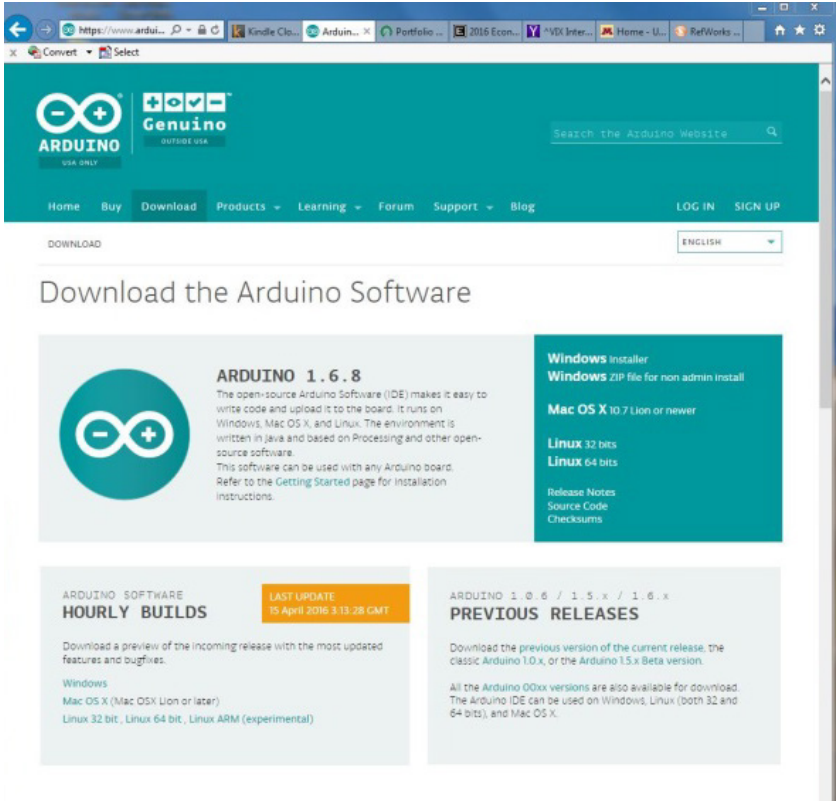
Software & Programming

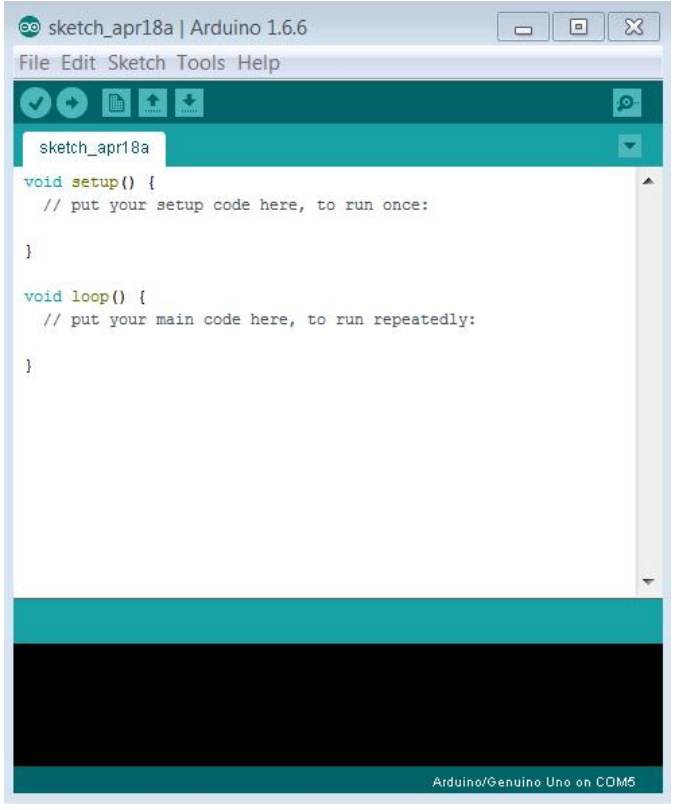
1. Software is the *smart* in your “smart device.”
2. An *algorithm* displayed as a *flowchart,* transforms your problem into various input, processing, decision and output steps
3. Lines of *code* are written to implement your algorithm.
4. Code may be written in assembly language and/or higher level languages such as C, C++, and C#.
5. A *compiler* converts your code into *machine language* that the *microcontroller* understands.
6. The *compiled code* is then *uploaded* into a board containing the microcontroller, memory and various interface circuits.
7. Errors are then fixed by *debugging*.
8. You may write your own code and/or incorporate code that has been written by others (“sketches”).

Integrated Development Environment (IDE)

- Editor – To write your code in C (.c) and/or assembly (.a) language. A finished program is called a “sketch.”
- Compiler – Turns your code into machine readable instructions or object files (.o). A Linker combines this code with the standard Arduino Libraries, producing a single hex file (.h).
- Means to Upload – Transferring the hex file to the Arduino board program memory. This is done via the USB or serial connection with the aid of the bootloader.
- Means to Run – *Executing* the Program
- Means to Debug – Finding & *Correcting Errors*

Programs are Written in C and/or Assembly Language





Arduino Bootloader & Firmware

- *Bootloader* is resident code that runs when the board is powered on or reset. It is programmed via the *ICSP* with a *programmer*.
- This code configures the board and *USB* port for your subsequent use and programming directly from your computer.
- You can instead remove the bootloader to save memory and program directly through the ICSP header.

Arduino Programming Components

1. **Structures**
   - A. Setup & loop
   - B. Control statements
   - C. Syntax
   - D. Arithmetic operators
   - E. Comparison operators
   - F. Bitwise operators
   - G. Compound operators

2. **Variables**
   - A. Constants
   - B. Data Types
   - C. Variable Scope
   - D. Qualifiers
   - E. Conversion
   - F. Utilities

3. **Functions**
   - A. Digital I/O
   - B. Analog I/O
   - C. Due & Zero only
   - D. Advanced I/O
   - E. Time
   - F. Math
   - G. Trigonometry
   - H. Characters
   - I. Random numbers
   - J. Bits and bytes
   - K. External interrupts
   - L. Interrupts
   - M. Communication
   - N. USB

Items in blue will be covered in this lecture, items in red in Arduino Part 2, and green in Arduino Part 3.

1. Structures: *Setup() Example*

1) Initialize variables
2) Assign pins
3) Runs once, after powerup or reset.

```
void setup()
{
   int buttonPin = 3;  // initialize buttonPin
   pinMode(buttonPin), INPUT);
                        //assign pin 3 to be an input
}

void loop()
{
…
}
```

Loop() Example

1. Occurs after setup.
2. Loops consecutively
3. Initialize variables
4. Assign pins
5. Runs once, after powerup or reset.
6. Variations: if, if-else; if-else-if

```
int buttonPin = 3;
void setup()
{
   Serial.begin(9600) ;                  //serial baud rate
   pinMode(buttonPin), INPUT);   //assign 3 to be an
                                                 input
}
void loop()
{
   if(digitalRead(buttonPin) == HIGH)  {
        Serial.write('H');
      }
   else {
     serial.write('L');
      }
   delay(1000);
 } //void loop
```

Control Statements

- Loop Statements
  - For
  - While
  - Do-While
- Decision Statements
  - Break and Continue
  - If
  - If-Else, if-else-if
  - Switch-Case
- Directional
  - Goto
  - Return

Arrays…

1. The first element is indexed with zero, e.g. a[3] has 3 elements, a[0], a[1], and a[2].
2. Declare as usual, e.g. int a[3], float a[3], and char a[3].
3. Initialize: int a[3] = {2, 6, 1}.
4. Ok to initialize using a “for” loop.
5. If the number of elements is not stated, the initialization will determine it, e.g. int a[ ] = {2, 6, 1} – elements will be three.
6. Arrays may be multidimensional, e.g. a[3, 5].
7. Two dimensional (rows and columns) can also be written, e.g. int M[4] [5] *(remember there is a zero row and column).*
8. Number of elements may be determined by variable – in which case range check first.

"For" Statement (a Loop)…

Statement Format

```
for (initialization; condition; increment)
   {program statement(s);}
```

Example – What is the value of the a[49] element?

Example Code

```
…
int a[100];
for (int n = 0; n < 100; n = n + 1)
 {
   a[n] = n * 2;
  }
…
```

"While" Statement (a Loop)…

while (*expression – a boolean that is true or false*)
{program statement(s);}

Example – What is the value of a[30] element?

```
…
int a[100];
int n = 0;
while (n < 100) {
    a[n] = n * 3;
    n = n + 1;  // Could also use
                "++n"
   }
…
```

"Do-while" Statement (a Loop)…

```
do
   {program statement(s)}
while (test condition);
```

Example – What is the value of a[75] element?

```
…
int a[100];
int n = 0;
do {
      a[n] = n * 4;
      n = n + 1;
  }
while n < 100;
…
```

"If" Statement (a Decision)…

if (*expression)*
  {program statement(s);}

Example – What is the value of n?

```
…
int a = 4, n = 0;
if a <= 5 {
   n = n + 50;
 }
…
```

"If-Else" Statement (a Decision)…

```
if (expression)
   {program statement(s);}
else
   {program statement(s)};
```

Example – What is the value of n?

```
…
int a = 10, n = 0;
if a <= 5 {
   n = n + 50;
 }
else {
   n = n  + 25;
 }
…
```

“Switch – Case” Statement (a Decision…)

```
switch (expression)
{
   case label1:
          program statements;
          break;
   case label2:
          program statements;
          break;
  default:
          program statements;
           break;
}
```

For example:

```
int a;
Bool buy;
…
a = 2;
switch (a)
{
   case 1:                    // if a =1
            buy = true;
             break;
   case 2:                     // if a =2
            buy = false;
             break;
}
…
```

Syntax

;	Used to end a statement
{ }	Enclose statements, keep balanced
//	Start comment until end of line
/* … */	Multi-line comment
#define	Assigning a value to a constant name Follows C rules and no semicolon afterwards Use *const type variable* = *value* (e.g. const float pi = 3.14) when able instead.
#include	To include outside libraries

Arithmetic & Boolean Operators

=	assignment operator
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

&&	and
||	or
!	not

Comparison & Pointer Operators

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

*	dereference
&	reference

2. Variables: *Constants*

1. true | false (typed in lower case)
    1. false is defined as zero
    2. true is defined as one, or any boolean test of an integer that is non-zero.
2. Integer constants:

• Decimal	123	
• Binary	B11110000	(leading B)
• Octal	0173	(leading zero)
• Hexadecimal	0x7B	(leading 0x)

5. Floating point constants:

Constant	Evaluates to	Also
10.0	10	
2.34E5	2.34 * 10^5	234000
67e-5	67.0 * 10^-5	.00067

Data Types for the Arduino Uno

1. bool – (8 bit) – one of two values, true or false.
2. boolean (8 bit) – non-standard type alias for bool.
3. byte – 8 bit unsigned number, 0-255.
4. char (8 bit) - A data type used to store a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC"). Each character has an ASCII numeric value, 0 to 255.
5. word (16 bit) - unsigned number from 0-65,535 (1 word = 2 bytes) (or 2^{16}-1).

https://learn.sparkfun.com/tutorials/data-types-in-arduino

6. int (16 bit) – Integers are the primary data type for number storage. Signed number from -32,768 to 32,767 (or $-(2^{15}-1)$ to 2^{15}). For negative numbers the highest bit designates "sign," while the rest of the number is inverted – the *compliment*.
7. unsigned int (16 bit) – Unsigned integer from 0 to 4,294,967,295 or $(2^{32} - 1)$.
8. long (32 bit) - signed number from -2,147,483,648 to 2,147,483,647 (or $-(2^{31}-1)$ to 2^{31}). If doing math with integers, at least one of the numbers must be followed by an L, forcing it to be a long.

9. unsigned long (32 bit) - unsigned number from 0 to 4,294,967,295 (or 2^{32}-1). The most common usage of this is to store the result of the *millis()* function, which returns the number of milliseconds the current code has been running.
10. float (32 bit or 4 bytes) – same as double - signed number from minus 3.4028235E38 to positive 3.4028235E38. Floats have 6-7 decimal digits of precision. Results are not exact.

https://learn.sparkfun.com/tutorials/data-types-in-arduino

Conversion

- char() - Converts a value to the char data type.
- byte() - Converts a value to the byte data type.
- int() - Converts a value to the int data type.
- word() - Converts a value to the word data type or creates a word from two bytes.
- long() - Converts a value to the long data type.
- float() - Converts a value to the float data type.

Variable Scope & Qualifiers

1. A global variable is one that can be *seen* by every function in a program.
   - Local variables are only visible to the function in which they are declared.
   - In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc.), is a global variable.
   - "For " loop variables are local.
2. Static - the static keyword is used to create variables that are visible to only one function.

Summary

- Arduino Uno, sensors and actuator examples.
- Using an IDE, programs are typically written in C and assembly language, compiled, linked with libraries and uploaded onto the Arduino board memory as hexadecimal code.
- Structures, variables and functions comprise an embedded program.
- Quiz